IN THE SPECIFICATION:

Please replace the paragraph on page 11, line 20, through page 12, line 5, with the following paragraph:

In the depicted example, local area network (LAN) adapter **260**, SCSI host bus adapter **262**, and expansion bus interface **264** are connected to PCI local bus **256** by direct component connection. In contrast, audio adapter **266**, graphics adapter **268**, and audio/video adapter (A/V) **269** are connected to PCI local bus ~~266~~ 256 by add-in boards inserted into expansion slots. Expansion bus interface **264** provides a connection for a keyboard and mouse adapter **270**, modem **272**, and additional memory **274**. SCSI host bus adapter **262** provides a connection for hard disk drive **276**, tape drive **278**, ~~and~~ CD-ROM **280**, and DVD **282** in the depicted example. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors.

A1

Please replace the paragraphs on page 1, line 11, through page 3, line 8, with the following paragraphs:

The use of platform independent code, such as ~~Java~~ JAVA, has increased with increase usage of the Internet. "JAVA" is a trademark of Sun Microsystems, Inc. This is because the Internet provides information, services, and computer programs to millions of client devices which may be configured in any number of different ways. Because it is rather impractical to require all client devices to adhere to a particular configuration, platform independent code provides a solution for allowing computer programs to execute properly on virtually all client devices, independent of the particular configuration of the client device.

A2

~~Java~~ JAVA is a programming language from Sun ~~Systems~~ Microsystems, Inc. that is designed for Internet (World Wide Web) and intranet applications. ~~Java~~ JAVA programs can be called from within HTML documents or launched stand alone. ~~Java~~ JAVA is an interpreted language that uses an intermediate language. The source code of a Java JAVA program is compiled into "byte code," which cannot be run by itself. The byte code must be converted into machine code at runtime.

Upon finding a ~~Java~~ JAVA applet, the Web browser on the client device switches to its ~~Java~~ JAVA interpreter, i.e. the ~~Java~~ JAVA Virtual Machine (JVM), which translates the byte code into machine code and runs it. This means ~~Java~~ JAVA programs are not dependent on any specific hardware and will run in any computer with the ~~Java~~ JAVA Virtual Machine.

While the Java JAVA code is platform independent, often ~~Java~~ JAVA applets will need native code, such as dynamically linked library files (.dll files), in order for the ~~Java~~ JAVA code to be executed correctly on a particular client device. These native code files are typically downloaded when the executed ~~Java~~ JAVA code indicates that a native code file is required.

A2

~~Java~~ JAVA applets and applications are routinely downloaded from servers to client devices over the Internet. During transmission of these ~~Java~~ JAVA applets and applications, it is possible that random corruption may occur such that the ~~Java~~ JAVA code that is received at the client device is not the same as the Java JAVA code sent by the server. More troublesome is the possibility of interception by a third party who may purposefully corrupt the ~~Java~~ JAVA code, e.g., by inserting a virus or the like.

Presently, known Java JAVA Application Program Interfaces (API) allow for some ability to check data integrity of ~~Java~~ JAVA code through the generation of digital signatures, e.g. through a one-way hash function or the like. However, currently, there is no API which allows for authentication of native code that is needed by the Java JAVA code. In other words, while the Java JAVA code may be authenticated as having not been corrupted during transmission from a server to the client device, the native code cannot be authenticated in this way.

Please replace the brief description for Figure 3B on page 6, lines 18-19, with the following text:

A3

Figure 3B is an exemplary block diagram of a ~~Java~~ JAVA Virtual Machine (JVM) according to the present invention;

Please replace the paragraphs on page 8, line 25, through page 9, line 23, with the following paragraphs:

The term "automatically authenticated code" as it is used in the present disclosure is meant to refer to code that is automatically authenticated through an existing mechanism, such as a virtual machine or the like, of a client device. The "automatically authenticated code" is automatically authenticated when received by the client device. An example of "automatically authenticated code" is compiled ~~Java~~ JAVA code that is automatically authenticated by the ~~Java~~ JAVA Virtual Machine (JVM) when received at the client device.

A4

The term "unauthenticated code" as it is used in the present disclosure is meant to refer to code that is not automatically authenticated by existing mechanisms when received by the client device. An example of "unauthenticated code" is native code that may be downloaded to a client device when necessary for proper execution of compiled Java JAVA code. The present invention provides a mechanism by which this "unauthenticated code" can be authenticated when downloaded for use with associated automatically authenticated code.

In the preferred embodiments of the present invention, as described hereafter, the "automatically authenticated code" will be assumed to be compiled ~~Java~~ JAVA code and the "unauthenticate code" will be assumed to be native code, for purposes of illustration of the features of the present invention. However, one of ordinary skill in the art should appreciate that the present invention is equally applicable to any type of automatically authenticated and unauthenticated code.

Please replace the paragraph on page 12, line 6-22 with the following paragraph:

An operating system runs on processor 252 and is used to coordinate and provide control of various components within data processing system 250 in **Figure 2B**. The operating system may be a commercially available operating system such as ~~Java~~ JAVA OS or OS/2, which are available from International Business Machines Corporation. Java JAVA OS is loaded from a server on a network to a network client and supports Java JAVA programs and applets. An object oriented programming system, such as Java JAVA, may run in conjunction with the operating system and may provide calls to the

A5

operating system from Java JAVA programs or applications executing on data processing system **250**. Instructions for the operating system, the object-oriented operating system, and applications or programs are located on storage devices, such as hard disk drive **276** and may be loaded into main memory **254** for execution by processor **252**. Hard disk drives are often absent and memory is constrained when data processing system **250** is used as a network client.

A5

Please replace the paragraphs on page 13, line 5, through page 18, line 3, with the following paragraphs:

The present invention provides an apparatus and method for ensuring the data integrity of unauthenticated code downloaded to a client device over a network. Although the present invention may operate on a variety of computer platforms and operating systems, it may also operate within a ~~Java~~ JAVA runtime environment. Hence, the present invention may operate in conjunction with a ~~Java~~ JAVA Virtual Machine (JVM) yet within the boundaries of a JVM as defined by ~~Java~~ JAVA standard specifications. In order to provide a context for the present invention, portions of the operation of a JVM according to ~~Java~~ JAVA specifications are herein described.

With reference now to **Figure 3A**, a block diagram illustrates the relationship of software components operating within a computer system that may implement the present invention. ~~Java-based~~ JAVA-based system **300** contains platform specific operating system **302** that provides hardware and system support to software executing on a specific hardware platform. JVM **304** is one software application that may execute in conjunction with the operating system. JVM **304** provides a ~~Java~~ JAVA run-time environment with the ability to execute ~~Java~~ JAVA application or applet **306**, which is a program, servlet, or software component written in the ~~Java~~ JAVA programming language. The computer system in which JVM **304** operates may be similar to data processing system **200** or computer **250** described above. However, JVM **304** may be implemented in dedicated hardware on a so-called ~~Java~~ JAVA chip, ~~Java-on-silicon~~ JAVA-on-silicon, or ~~Java~~ JAVA processor with an embedded ~~picoJava~~ picoJAVA core.

A6

At the center of a ~~Java~~ JAVA run-time environment is the JVM, which supports all aspects of ~~Java's~~ JAVA's environment, including its architecture, security features,

mobility across networks, and platform independence. The JVM is a virtual computer, i.e. a computer that is specified abstractly. The specification defines certain features that every JVM must implement, with some range of design choices that may depend upon the platform on which the JVM is designed to execute. For example, all JVMs must execute ~~Java~~ JAVA bytecodes and may use a range of techniques to execute the instructions represented by the bytecodes. A JVM may be implemented completely in software or somewhat in hardware. This flexibility allows different JVMs to be designed for mainframe computers and PDAs.

The JVM is the name of a virtual computer component that actually executes ~~Java~~ JAVA programs. ~~Java~~ JAVA programs are not run directly by the central processor but instead by the JVM, which is itself a piece of software running on the processor. The JVM allows ~~Java~~ JAVA programs to be executed on a different platform as opposed to only the one platform for which the code was compiled. ~~Java~~ JAVA programs are compiled for the JVM. In this manner, Java JAVA is able to support applications for many types of data processing systems, which may contain a variety of central processing units and operating systems architectures. To enable a ~~Java~~ JAVA application to execute on different types of data processing systems, a compiler typically generates an architecture-neutral file format – the compiled code is executable on many processors, given the presence of the ~~Java~~ JAVA run-time system.

A6

The ~~Java~~ JAVA compiler generates bytecode instructions that are nonspecific to a particular computer architecture. A bytecode is a machine independent code generated by the Java JAVA compiler and executed by a Java JAVA interpreter. A ~~Java~~ JAVA interpreter is part of the JVM that alternately decodes and interprets a bytecode or bytecodes. These bytecode instructions are designed to be easy to interpret on any computer and easily translated on the fly into native machine code.

A JVM must load class files and execute the bytecodes within them. The JVM contains a class loader, which loads class files from an application and the class files from the Java JAVA application programming interfaces (APIs) which are needed by the application. The execution engine that executes the bytecodes may vary across platforms and implementations.

One type of software-based execution engine is a just-in-time (JIT) compiler. With this type of execution, the bytecodes of a method are compiled to native machine code upon successful fulfillment of some type of criteria for "jitting" a method. The native machine code for the method is then cached and reused upon the next invocation of the method. The execution engine may also be implemented in hardware and embedded on a chip so that the ~~Java~~ JAVA bytecodes are executed natively. JVMs usually interpret bytecodes, but JVMs may also use other techniques, such as just-in-time compiling, to execute bytecodes.

When an application is executed on a JVM that is implemented in software on a platform-specific operating system, a ~~Java~~ JAVA application may interact with the host operating system by invoking native method, i.e. native code. A ~~Java~~ JAVA method is written in the ~~Java~~ JAVA language, compiled to bytecodes, and stored in class files. A native method is written in some other language and compiled to the native machine code of a particular processor. Native methods are stored in a dynamically linked library whose exact form is platform specific.

A6

With reference now to **Figure 3B**, a block diagram of a JVM is depicted in accordance with a preferred embodiment of the present invention. JVM **350** includes a class loader subsystem **352**, which is a mechanism for loading types, such as classes and interfaces, given fully qualified names. JVM **350** also contains runtime data areas **354**, execution engine **356**, native method interface **358**, and memory management **374**. Execution engine **356** is a mechanism for executing instructions contained in the methods of classes loaded by class loader subsystem **352**. Execution engine **356** may be, for example, ~~Java~~ JAVA interpreter **362** or just-in-time compiler **360**. Native method interface **358** allows access to resources in the underlying operating system. Native method interface **358** may be, for example, a ~~Java~~ JAVA native interface.

Runtime data areas **354** contain native method stacks **364**, ~~Java~~ JAVA stacks **366**, PC registers **368**, method area **370**, and heap **372**. These different data areas represent the organization of memory needed by JVM **350** to execute a program.

~~Java~~ JAVA stacks **366** are used to store the state of ~~Java~~ JAVA method invocations. When a new thread is launched, the JVM creates a new ~~Java~~ JAVA stack for the thread. The JVM performs only two operations directly on ~~Java~~ JAVA stacks: it

pushes and pops frames. A thread's Java JAVA stack stores the state of ~~Java~~ JAVA method invocations for the thread. The state of a Java JAVA method invocation includes its local variables, the parameters with which it was invoked, its return value, if any, and intermediate calculations. ~~Java~~ JAVA stacks are composed of stack frames. A stack frame contains the state of a single Java JAVA method invocation. When a thread invokes a method, the JVM pushes a new frame onto the ~~Java~~ JAVA stack of the thread. When the method completes, the JVM pops the frame for that method and discards it.

The JVM does not have any registers for holding intermediate values; any Java JAVA instruction that requires or produces an intermediate value uses the stack for holding the intermediate values. In this manner, the Java JAVA instruction set is well-defined for a variety of platform architectures.

A6

PC registers 368 are used to indicate the next instruction to be executed. Each instantiated thread gets its own pc register (program counter) and ~~Java~~ JAVA stack. If the thread is executing a JVM method, the value of the pc register indicates the next instruction to execute. If the thread is executing a native method, then the contents of the pc register are undefined.

Native method stacks 364 store the state of invocations of native methods. The state of native method invocations is stored in an implementation-dependent way in native method stacks, registers, or other implementation-dependent memory areas. In some JVM implementations, native method stacks 364 and ~~Java~~ JAVA stacks 366 are combined.

Please replace the paragraphs on page 18, line 21, through page 19, line 9, with the following paragraphs:

While the present invention is applicable to any system in which automatically authenticated and unauthenticated code are transmitted from a server device to a client device, the preferred embodiments of the present invention will be described in terms of a ~~Java~~ JAVA execution environment. Thus, the embodiments of the present invention will be explained in terms of signed ~~Java~~ JAVA code, unauthenticated native code, ~~Java~~ JAVA Virtual Machines, and the like. It should be appreciated by those of ordinary skill

A7

in the art that the present invention is equally applicable to other similar execution environments.

A7

As mentioned above, when a client device requests an application from a trusted web server, such as a ~~Java~~ JAVA applet or application, the application is downloaded to the client device as compiled ~~Java~~ JAVA code. The compiled Java JAVA code includes an electronic signature which is used to verify the integrity of the application data received by the client device.

Please replace the paragraph on page 19, line 27, through page 20, line 6, with the following paragraph:

A8

When the application is run on the client device, the application may require additional data files to be downloaded for execution on the particular client device. For example, if the application is a Java JAVA applet or application, the ~~Java~~ JAVA applet may need native methods, e.g., dynamically linked library (.dll) files, so that the ~~Java~~ JAVA applet can be properly executed on the client device.